ESTRUCTURAS DE DATOS PARA LA ORGANIZACIÓN Y BUSQUEDA EFICIENTE DE ARCHIVOS

Isabela Muriel Roldan
Universidad Eafit
Colombia
imurielr@eafit.edu.co

Mateo Flórez Restrepo
Universidad Eafit
Colombia
mflorezr@eafit.edu.co

Juan Pablo Castaño Duque
Universidad Eafit
Colombia
jpcastanod@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

RESUMEN:

El objetivo de este informe es analizar y solucionar el problema de la búsqueda eficiente de archivos y subdirectorios almacenados en un directorio. Lo que se busca es encontrar y utilizar una estructura de datos que permita guardar los archivos de manera organizada con el fin de que, a la hora de consultarlos el sistema los encuentre en el menor tiempo posible.

La solución a este problema es de gran importancia debido a que a medida que la tecnología avanza se necesita almacenar cada vez mayor número de información y si esta no es guardada de manera organizada se pueden llegar a generar grandes molestias debido a la poca capacidad de memoria y la ineficiencia a la hora de consultar los archivos. Existen varios problemas similares al que se plantea en este informe y algunos de estos serán analizados con el propósito de hallar una solución más efectiva.

1. INTRODUCCIÓN:

En el día a día, en cuanto a la computación y tecnologías respecta, se busca un progreso constante para que cada vez las cosas sean más útiles, más optimas y de un uso mucho más sencillo para el usuario, lo que trae como consecuencia que los programas se vayan haciendo obsoletos a medida que las tecnologías avanzan.

Un gran ejemplo de lo mencionado anteriormente son los sistemas de archivos, este es el encargado de almacenar los archivos de un sistema operativo, pero como todo programa tiene sus limitaciones, una de ellas es la cantidad de archivos que puede almacenar ya que lo que en un tiempo pudo ser una cifra exorbitante hoy es una cifra insuficiente para la cantidad de datos que se manejan, por lo tanto se necesita encontrar una manera eficiente de almacenar estos archivos e incluir otras funciones que sean útiles para el mismo, tales como la búsqueda. Ese es el objetivo que se tiene en este proyecto.

2. PROBLEMA

El problema al cual nos enfrentamos se basa en crear a través de una estructura de datos un sistema de archivos eficiente en cuanto a las especificaciones de un cliente para encontrar fácilmente todos aquellos archivos, ficheros y subdirectorios que se encuentren listados dentro un directorio.

Resolver este problema sería un avance en cuanto a los sistemas de archivos y almacenamiento que maneja un sistema operativo, sobre todo para aquellas grandes empresas que trabajan una gran cantidad de información importante.

3. TRABAJOS RELACIONADOS

3.1. Tablas hash

Las tablas hash son estructuras de datos utilizadas para almacenar gran cantidad de datos que luego requieran ser buscados e insertados (permite almacenar y recuperar) en un sistema de archivos muy eficiente. Una tabla hash almacena un conjunto de pares “(clave, valor)”. La clave es única para cada elemento de la tabla y es el dato que se utiliza para buscar un determinado valor. Un contenedor asociativo.

La implementación de una tabla hash está basada en los siguientes elementos:

- Una tabla de un tamaño razonable para almacenar los pares (clave, valor).
- Una función “hash” que recibe la clave y devuelve un índice para acceder a una posición de la tabla.
- Un procedimiento para tratar los casos en los que la función anterior devuelve el mismo índice para dos claves distintas. Esta situación se conoce con el nombre de colisión.

Las posibles implementaciones de cada uno de estos tres elementos se traducen en una infinidad de formas de implementar una tabla hash

Se debe tener cuidado con el tamaño que requiere una tabla hash de acuerdo al tipo de aplicación y su implementación, ya que si es muy pequeña puede llegar a convertirse en una lista encadenada, en cuyo caso la búsqueda puede ser muy ineficiente ya que se debe pasar por cada valor, y si por el contrario la tabla es enorme, debido a las posiciones vacías que quedaran la cantidad de memoria se malgasta innecesariamente.

3.2. Árbol Rojo-Negro

El problema con algunas estructuras de datos binarias es que en la mayoría de los casos un lado del árbol queda más sobrecargado que el otro, lo que hace que, a la hora de realizar el proceso de búsqueda, se alargue el camino a recorrer para hallar los datos y el tiempo de espera es mucho mayor. El árbol rojo- negro consiste en un árbol de búsqueda binaria, esto significa que agrupa los valores en dos grupos, a la izquierda los menores y a la derecha los mayores, pero la diferencia entre este y otros árboles binarios es que a este se le incorpora un bit extra de almacenamiento por nodo, lo que permite determinar si un nodo es rojo o negro según las propiedades de esta estructura. Con ayuda de dichas propiedades y el color de cada nodo, se determina un orden de los datos que va cambiando a media que se agrega nueva información, de manera que el árbol queda equilibrado y acorta el camino a la hora de buscar un dato determinado.

3.3. Árbol B:

Es un árbol de búsqueda que puede estar vacío o aquel cuyos nodos pueden tener varios hijos, existiendo una relación de orden entre ellos, tal como muestra la Figura 1.

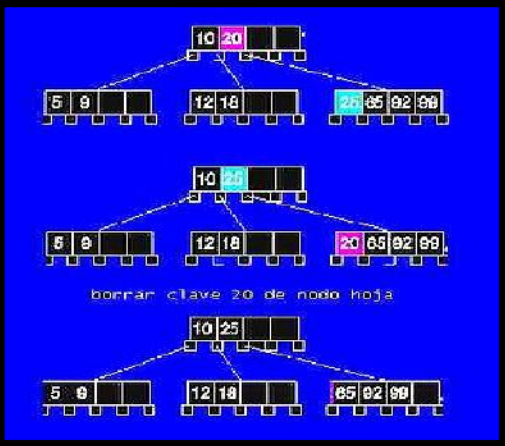

Figura 1: Árbol B. Ejemplo.

Este árbol tiene una organización ascendente, siempre el número a la izquierda es menor que el de la derecha.

Propiedades:

- Es un árbol multicaminos.
- Todas sus hojas están en el mismo nivel.
- Los datos se llaman claves y se almacenan en páginas.
- “M” es el grado del árbol y es el máximo número de hijos que puede tener cualquiera página.
- “M-1” es el número de claves que puede tener cualquier página.
- “M-1/2” Es el número mínimo de claves que puede tener cualquier página.

Procesos que puede realizar el árbol:

Los arboles B, realizan funciones definidas por algoritmos, las principales son:

Búsqueda:

1. Situarse en el nodo raíz.
2. (*) Comprobar si contiene la clave a buscar.
    1. Encontrada fin de procedimiento.
    2. No encontrada:
        1. Si es hoja no existe la clave.
        2. En otro caso el nodo actual es el hijo que corresponde:
            1. La clave a buscar k < k1: hijo izquierdo.
            2. La clave a buscar k > ki y k < ki+1 hijo iésimo.
            3. Volver a paso 2(*).

Inserción:

1. Realizando una búsqueda en el árbol, se halla el nodo hoja en el cual debería ubicarse el nuevo elemento.
2. Si el nodo hoja tiene menos elementos que el máximo número de elementos legales, entonces hay lugar para uno más. Inserte el nuevo elemento en el nodo, respetando el orden de los elementos.
3. De otra forma, el nodo debe ser dividido en dos nodos. La división se realiza de la siguiente manera:
    1. Se escoge el valor medio entre los elementos del nodo y el nuevo elemento.
    2. Los valores menores que el valor medio se colocan en el nuevo nodo izquierdo, y los valores mayores que el valor medio se colocan en el nuevo nodo derecho; el valor medio actúa como valor separador.
    3. El valor separador se debe colocar en el nodo padre, lo que puede provocar que el padre sea dividido en dos, y así sucesivamente.

3.4. Árbol B+

Es una estructura de datos tipo árbol que representa una colección de datos ordenados y que además permite la inserción y eliminación de manera eficiente de los elementos que se encuentran en la colección. A diferencia del Árbol B ya mencionado anteriormente y siendo una variante del mismo en un

árbol B+, toda la información se guarda en las hojas. Los nodos internos sólo contienen claves y punteros. Todas las hojas se encuentran en el mismo nivel, que corresponde al más bajo. Los nodos hoja se encuentran unidos entre sí como una lista enlazada para permitir principalmente recuperación en rango mediante búsqueda secuencias.

REFERENCIAS

1. Stalin, F. Árbol rojo negro la evolución de los árboles binarios de búsqueda. Retrieved August 10, 2017, from Computer science in esmeraldas. http://computerscienceesmeraldas.blogspot.com.co/2016/03/arbol-rojo-negro-la-evolucion-de-los.html.

2. Universidad Carlos III de Madrid. Tablas hash. Retrieved August 12, 2017, from Departamento de ingeniería telemática. http://www.it.uc3m.es/abel/as/MMC/M2/HashTable_es.html.

3. Wikipedia. Árbol-B. Retrieved August 11, 2017. https://es.wikipedia.org/wiki/Árbol-B.

4. Wikipedia. Árbol B+. Retrieved August 11, 2017. https://es.wikipedia.org/wiki/Árbol_B%2B.

5. Solis, S. Árboles B. Retrieved August 12, 2017, from youtube https://www.youtube.com/watch?v=EbiFITGh0rI